MEMORANDUM
RM-6241-ARPA
APRIL 1970

THE GRAIL RING STRUCTURE AND PRIMITIVES

T. O. Ellis, J. F. Heafner and W. L. Sibley

PREPARED FOR
ADVANCED RESEARCH PROJECTS AGENCY

The RAND Corporation
SANTA MONICA • CALIFORNIA

ARPA ORDER NO. 189-1

MEMORANDUM
RM-6241-ARPA
APRIL 1970

THE GRAIL RING STRUCTURE AND PRIMITIVES

T. O. Ellis, J. F. Heafner and W. L. Sibley

This research is supported by the Advanced Research Projects Agency under Contract No. DAHC15 67 C 0141. Views or conclusions contained in this study should not be interpreted as representing the official opinion or policy of Rand or of ARPA.

The RAND Corporation
1700 MAIN ST • SANTA MONICA • CALIFORNIA • 90406

PREFACE

This Memorandum further documents the man-machine interaction studies sponsored by the Advanced Research Projects Agency. The data organization illustrates the needs of a typical on-line relational data file, such as might be found in a military command and control environment.

This Memorandum exemplifies techniques used in the implementation of one functional aspect of the GRAIL (Graphical Input Language) system. The GRAIL system techniques are now being used at RAND in a series of experiments designed for direct application to the Defense Intelligence Agency and for the Air Force Aeronautical Chart & Information Center. It is expected the techniques will also be useful to the Army Map Service and the Navy carrier force command-control and intelligence facility.

DoD agencies, military agencies and ARPA contractors enlisted in constructing interactive graphical systems may obtain detailed information on other functional aspects of GRAIL by contacting the Computer Systems Group, Computer Sciences Department, The RAND Corporation, 1700 Main Street, Santa Monica, California 90406.

SUMMARY

RING STRUCTURES

The disposition of data and available storage space described in this Memorandum is effected by the GRAIL system by a hierarchy of ring structures that represent the logic of both the user's program and the system itself. As their name implies, ring structures are continuous chains of elements compartmentalized in a structure space in such a fashion that any unused space is always compressed within a single area whose boundaries change with the number of elements in use at any given time.

This Memorandum describes five ring structure types:

1) The *system structure* describes space allocation and data disposition in the system's secondary area. It also contains information about the current user's file. When GRAIL is in use, a copy of the system structure exists in primary storage, and is updated in secondary storage whenever the operating environment changes.
2) The *files description structure* describes in a gross way all secondary storage space that may be allocated to the user's files; it resides at a fixed location in the user's secondary storage area. A copy is read into primary storage only to access or to manipulate a file.
3) The *file structure* describes space allocation within a user's file as well as the interrelationships, names, and secondary storage locations of labeled process definitions.

Part of the above three ring structure types is a *space allocation substructure* that describes the location of data sets and space available for use in secondary storage.

The two remaining ring structure types are the context and plane structures:

4) The *context structure* is a part of the internal organization of a labeled process definition. One such structure exists for each labeled process definition in a user's file. It contains information about labels (and their attributes) and the hierarchy of unlabeled process instances within the labeled process definitions. A copy of the process definition exists in a primary storage whenever a display frame of that process definition is being displayed.
5) The *plane structure*, one for each process definition, contains information about the connectivity of graphical elements in each display frame and the translation of parameters for each labeled process instance in the frame. It also provides information that permits an association between the virtual image and its internal logical representation. A copy of the plane representing the process being operated upon exists in primary storage.

PRIMITIVES

Primitives are remote code sequences that perform basic operations on all GRAIL ring structures. They obtain and release elements, locate elements, and modify the ring composition.

CONTENTS

PREFACE iii

SUMMARY v

FIGURES ix

Section
I. INTRODUCTION 1
Elements and Rings 1
Structure Space 4

II. THE GRAIL RING STRUCTURES 7
Space Allocation Substructure 7
System Structure 9
Files Description Structure 12
File Structure 12
Context Structure 15
Plane Structure 19

III. RING STRUCTURE PRIMITIVES 27
Functions of Primitives 27
Obtain and Release Primitives 27
Locate Attribute Primitives 28
Structure Modifying Primitives 31

REFERENCES 33

FIGURES

1. Ring Structure 2
2. Ring Structure 3
3. Ring Structure Pointers 5
4. An External Element 6
5. Space Allocation Substructure 8
6. Available Heads Element 10
7. System Structure 11
8. Files Description Structure 13
9. File Structure 14
10. Plane Structure 16
11. Context Structure 17
12. Plane Structure 21
13. Plane Structure 22
14. Plane Structure 23
15. Plane Structure 24
16. Plane Structure 25
17. Plane Structure 26

I. INTRODUCTION

Because representation and maintenance of the GRAIL system and user's program requires data far in excess of what can be accommodated in primary storage, the bulk of this information must be stored in secondary storage. Thus arises the necessity of monitoring the disposition of data and of all remaining available space.

In general, ring structures are used to represent the logic of both the user's program and the system itself. For example, they describe the connectivity of flow diagrams, and, less apparently, the space allocation of a user's program in secondary storage.

ELEMENTS AND RINGS

Rings are continuous chains of elements. Elements are either 8 or 16 bytes in length. Each 4-byte word of a 8-byte element and the first two words of a 16-byte element consist of a code and either a link or a datum. The last two words of a 16-byte element are data (see Fig. 1). The first bit of the code byte determines whether the following three bytes constitute a link or a datum. Links, which are relative addresses of other elements, are maintained by a group of ring structure primitives. All links are relative to the beginning of the space in which the ring structure resides.

The upper link is the *object* link and the lower is the *set link* (see Fig. 2). One or more elements linked through the object link make up an *object ring*. Similarly, one or more elements linked through the set link compose a *set ring*.

Basic operations on ring structures--such as finding, procuring, and releasing elements--are performed by ring structure primitives, written as a group of remote code sequences. A mechanism employed in these operations is an

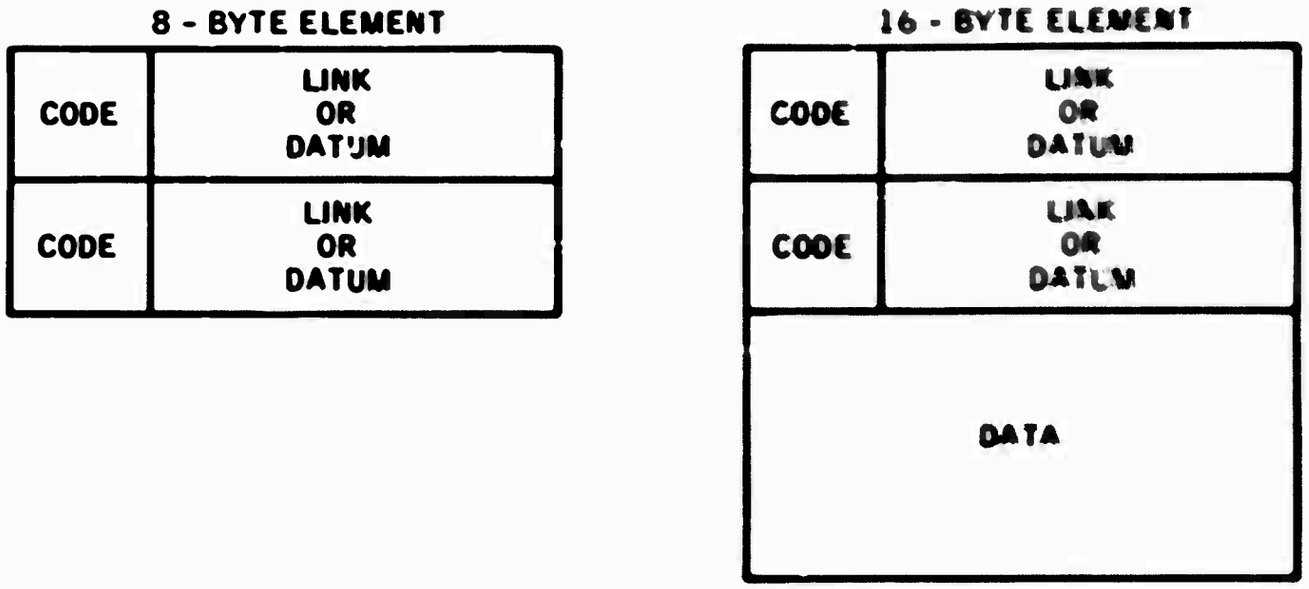


Fig. 1--Ring Structure

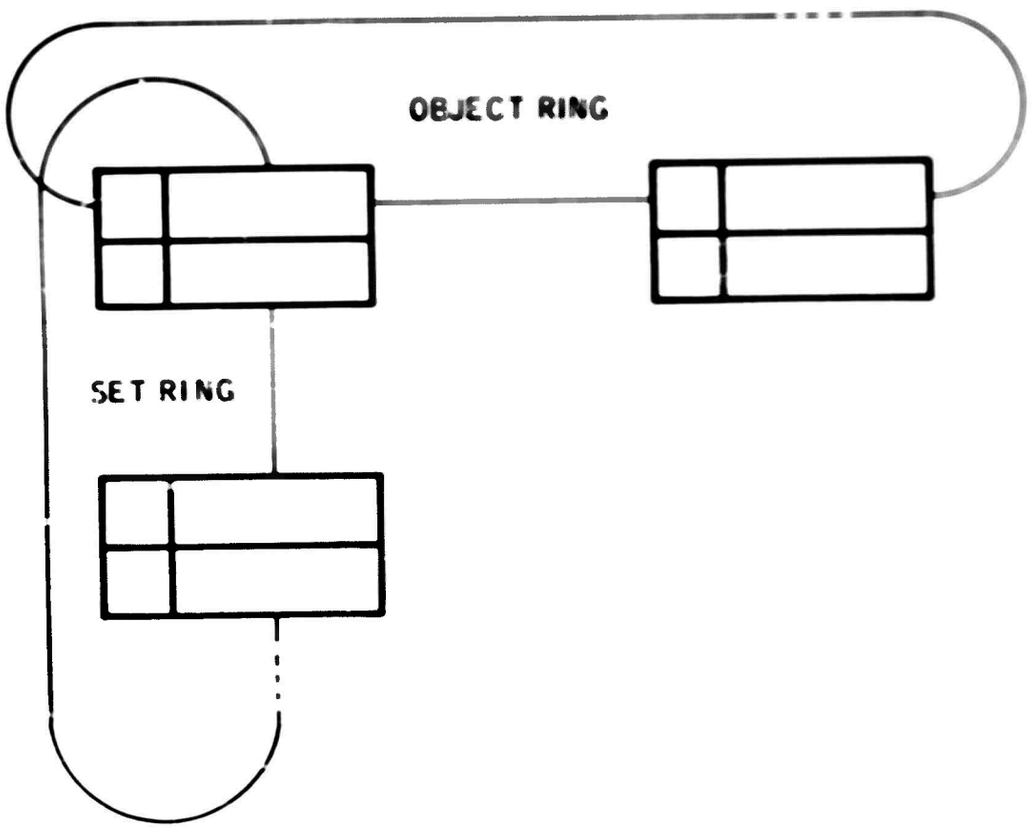


Fig. 2--Ring Structure

array of pointers (see Fig. 3) that delimit the boundaries of the structure's space. Though external to the structure space, the array normally resides in the same automatic storage that contains the structure space.

STRUCTURE SPACE

Rings of elements are compartmentalized in a structure space (see Fig. 3). The boundaries of the available space (see Fig. 3) change according to the number of elements in use at any given time. When an element of either length is requested, the appropriate element bounding the available space is supplied. When returned, the appropriate boundary element of the type in use is interchanged with the element to be returned. This operation keeps the elements in use separated from the available space.

To conserve secondary storage space, the structure space is compressed and only those elements in use are recorded.

Elements external to the structure space--for example, a Pseudo Command Channel Word (PCCW) in the table of PCCWs shown in Fig. 4--are permissible while the structure is in primary storage. The links of such external elements are automatically updated by the primitives. This provides a current reference to a (possibly) relocated ring.

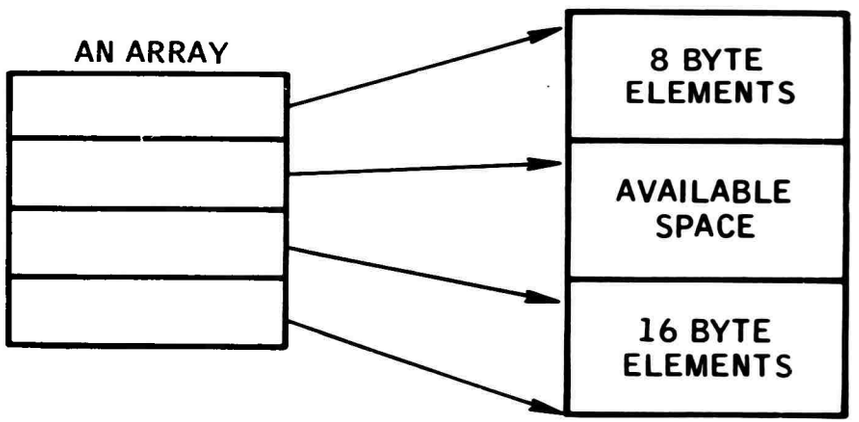


Fig. 3--Ring Structure Pointers

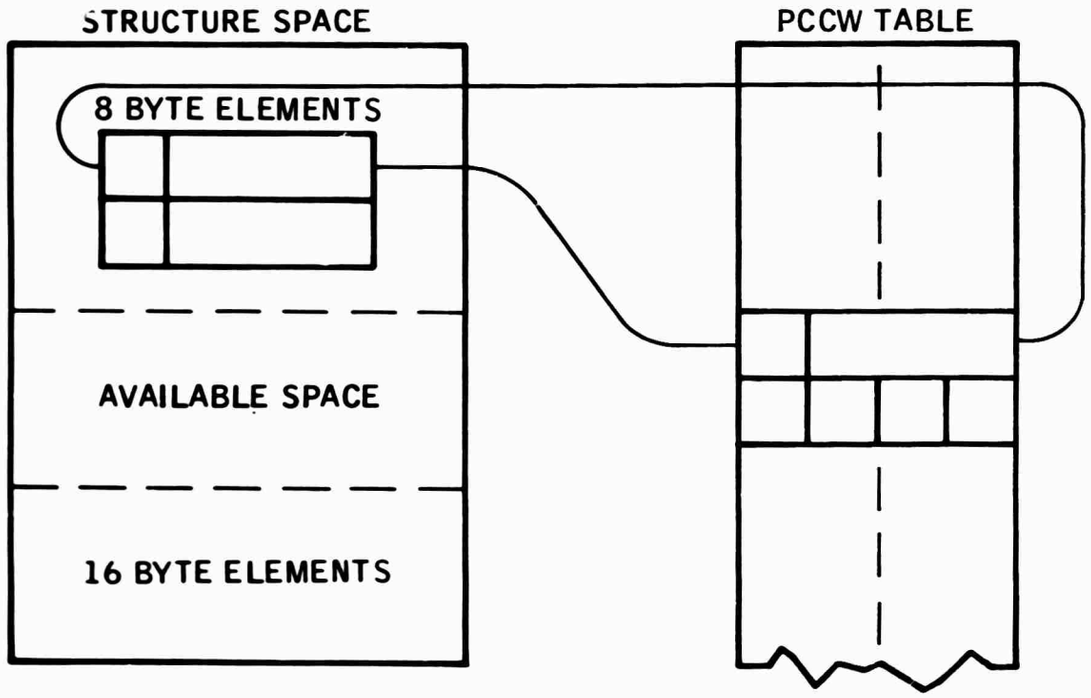


Fig. 4--An External Element

II. THE GRAIL RING STRUCTURES

There are five basic ring structure types: 1) the system structure, 2) the files description structure, 3) file structures, 4) context structures, and 5) plane structures.

SPACE ALLOCATION SUBSTRUCTURE

Common to the first three structure types enumerated above is a *space allocation substructure* (see Fig. 5) that describes the location of data sets within a particular area in secondary storage, and also space available for use.[†] It consists of three parts or rings: the *occupied* set ring, the *partially available heads* object ring, and the *available heads* object ring.

The space allocation substructure relates to a particular region of secondary storage space. Within this space are data sets. The location of some, but not all, of these data sets is given in the occupied set ring. ID is the internal identifier of a data set. C,H is the relative cylinder and head (i.e., the location) of that data set in secondary storage.[‡] A group of logical I/O system processes adds elements or deletes elements from the occupied set ring under program control when adding or deleting a data set. Optionally, they can pass the relative C,H to the invoking process for recording elsewhere. Thus, not all data sets in this area appear in the occupied set ring. Typically, static display frames and compiled read-only processes are recorded here; the location of dynamic display frames and ring structures are kept elsewhere.

[†]The secondary storage used was two IBM 2311 disk drives, one for the system and one for user's files.

[‡]The disk pack is divided into 200 cylinders, each cylinder containing 10 heads.

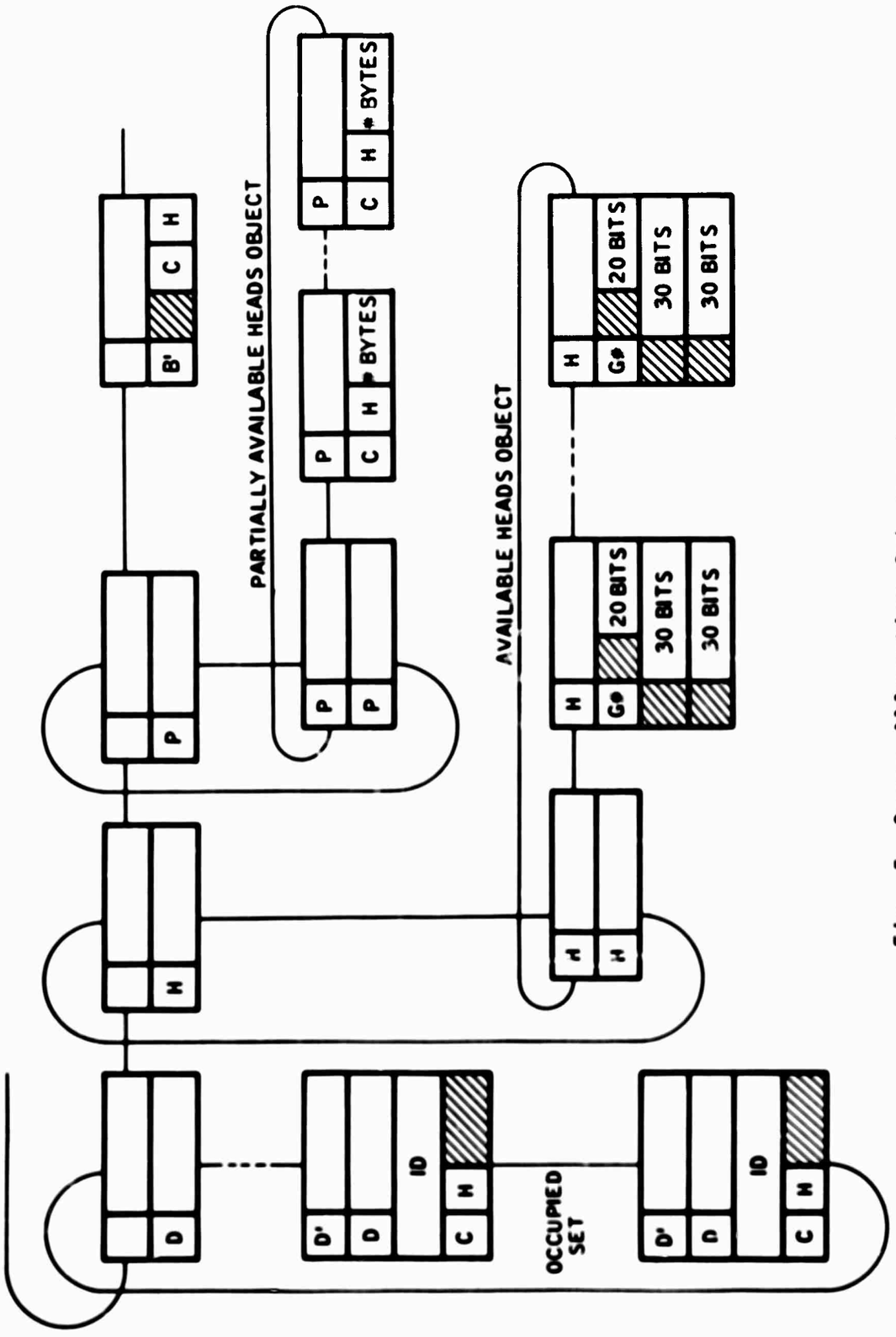


Fig. 5--Space Allocation Substructure

A disk head can exist in one of three different states: completely occupied, completely available, or partially occupied. The partially available object ring (see Fig. 5) describes the latter case, but does not relate to the data sets themselves, but only to the amount of free space available. In particular, the C,H of the element gives the relative cylinder and head location; the number of bytes in the element indicates the amount of available space in that head. This object ring also is maintained by the logical I/O processes.

The available heads object ring specifies the availability or nonavailability of each head within the space. (See Fig. 6 for details.) It is used when a completely available head is needed, and is maintained by the logical I/O processes.

Because the location of data sets in these three rings is relative, the absolute location is obtained (for purposes of reading and writing) by adding to it the base C,H in the element labeled "Base." Addresses are kept in relative form to facilitate, for example, replication of files; i.e., when a file is copied, only the base address of the copy need be updated.

SYSTEM STRUCTURE

The *system structure* (see Fig. 7) is a directory, describing space allocation for the system in secondary storage. It also contains information about the current or last-addressed user's file. Except for the Initial Program Load,† it is the only data set that resides at a fixed location in the system's secondary storage area. When GRAIL is in use, a copy of the system structure exists in primary storage. This copy is rewritten on secondary storage whenever the

†The Initial Program Load Record is the first 24 bytes of cylinder 0, head 0 on each disk pack.

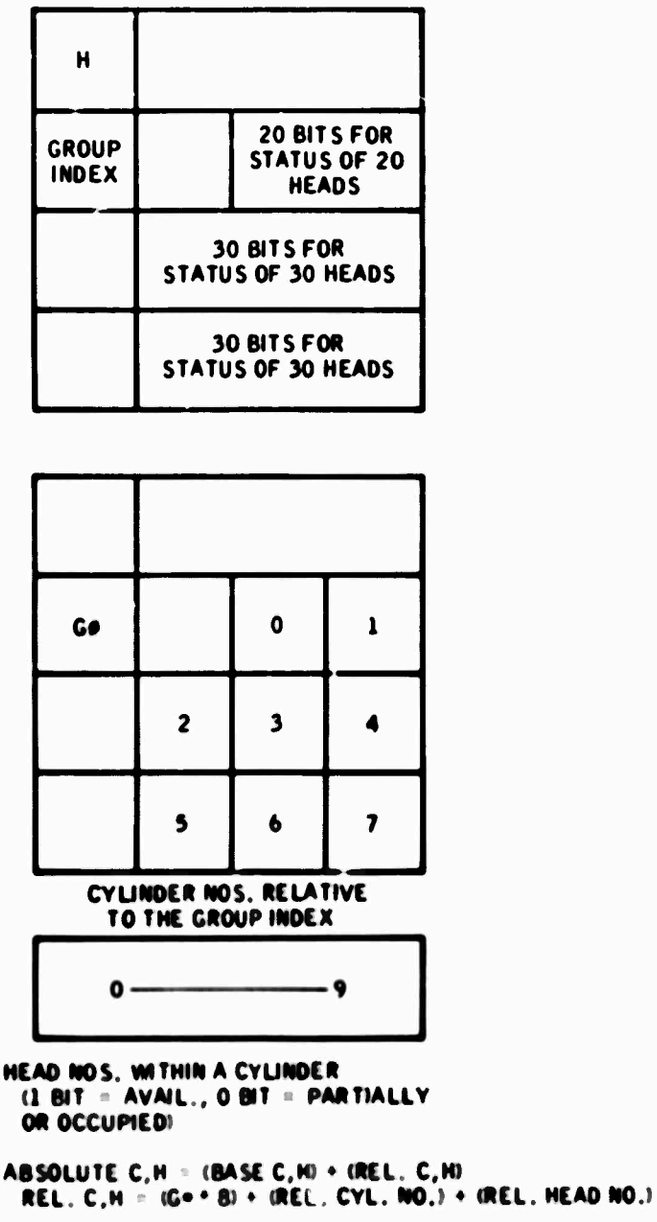


Fig. 6--Available Heads Element

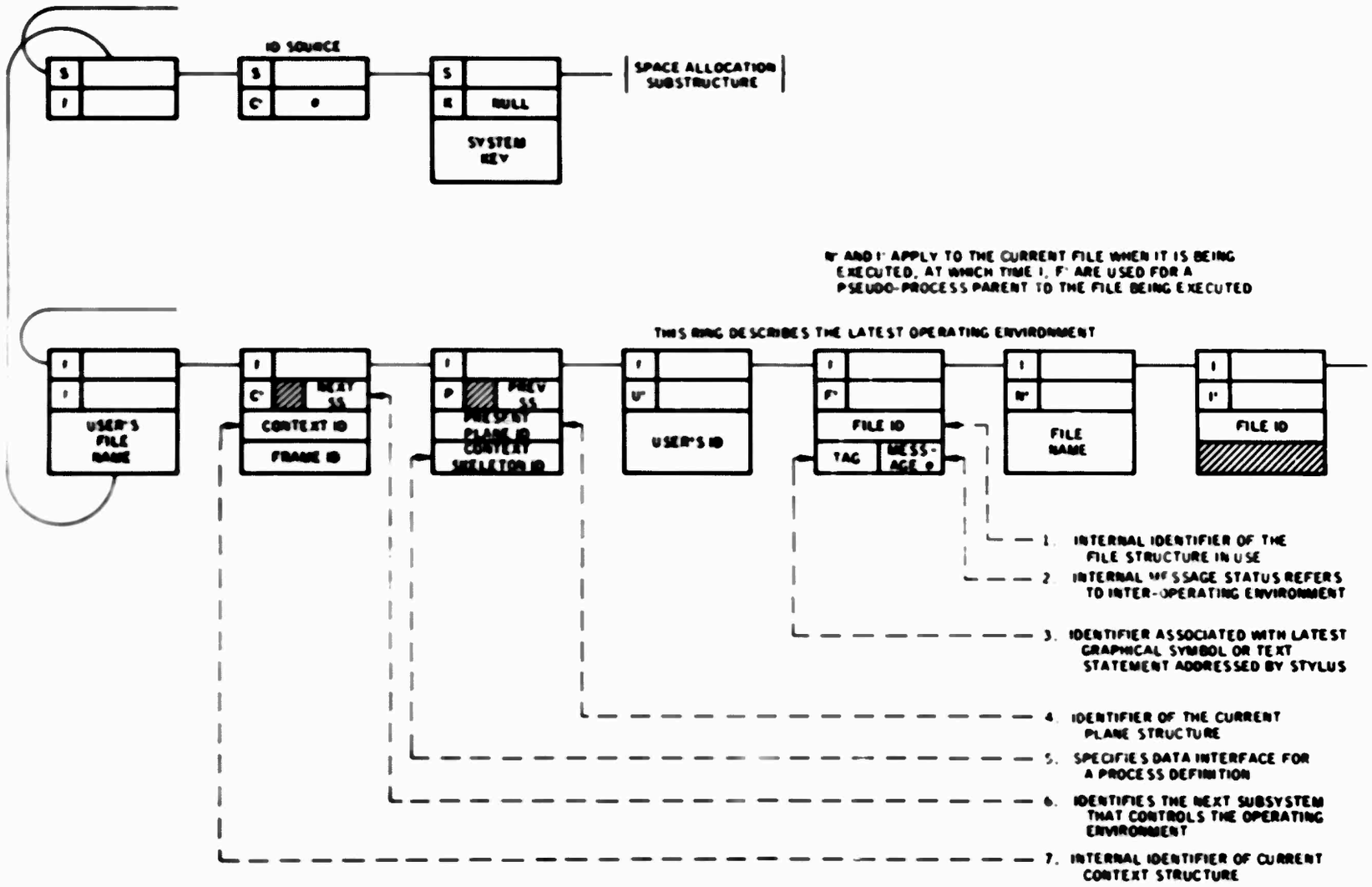


Fig. 7--System Structure

operating environment changes so that conditions can be re-established if there is a system failure.

The contents of the identification source element (see Fig. 7) are distributed as a *formal parameter* throughout the system, and are used as a source for internal identifiers of new data sets. The system key is a label identifying the current version of the system. Note: in Fig. 7, the previously described space allocation substructure is present. (See Fig. 8 for details of user's file information.)

FILES DESCRIPTION STRUCTURE

Similar to the system structure, which describes the system's secondary storage area, is the Files Description Structure (FDS), which describes the secondary storage area allocated to the users' files (see Fig. 8). It is the only data set in the users' secondary storage area that resides at a fixed location.† Unlike the copy of the system structure, which resides in primary storage continuously, a copy of the files description structure is read into primary storage only when there is need to access or manipulate some file that it describes.

Like the system key in the system structure, the version ID is the current version identifier. Again, the space allocation substructure is present. Additionally, the files description structure contains a set of *file* object rings, whose functions are labeled and described in Fig. 8. The type identifier is necessary because the log-on procedure and the file-accession mechanism of the GRAIL system have been adapted to other GRAIL-derived programs.

FILE STRUCTURE

The file structure (see Fig. 9) in the user's file describes space allocation within that file, and also

---

†An IPL record may reside at 0,0 on this pack as well.

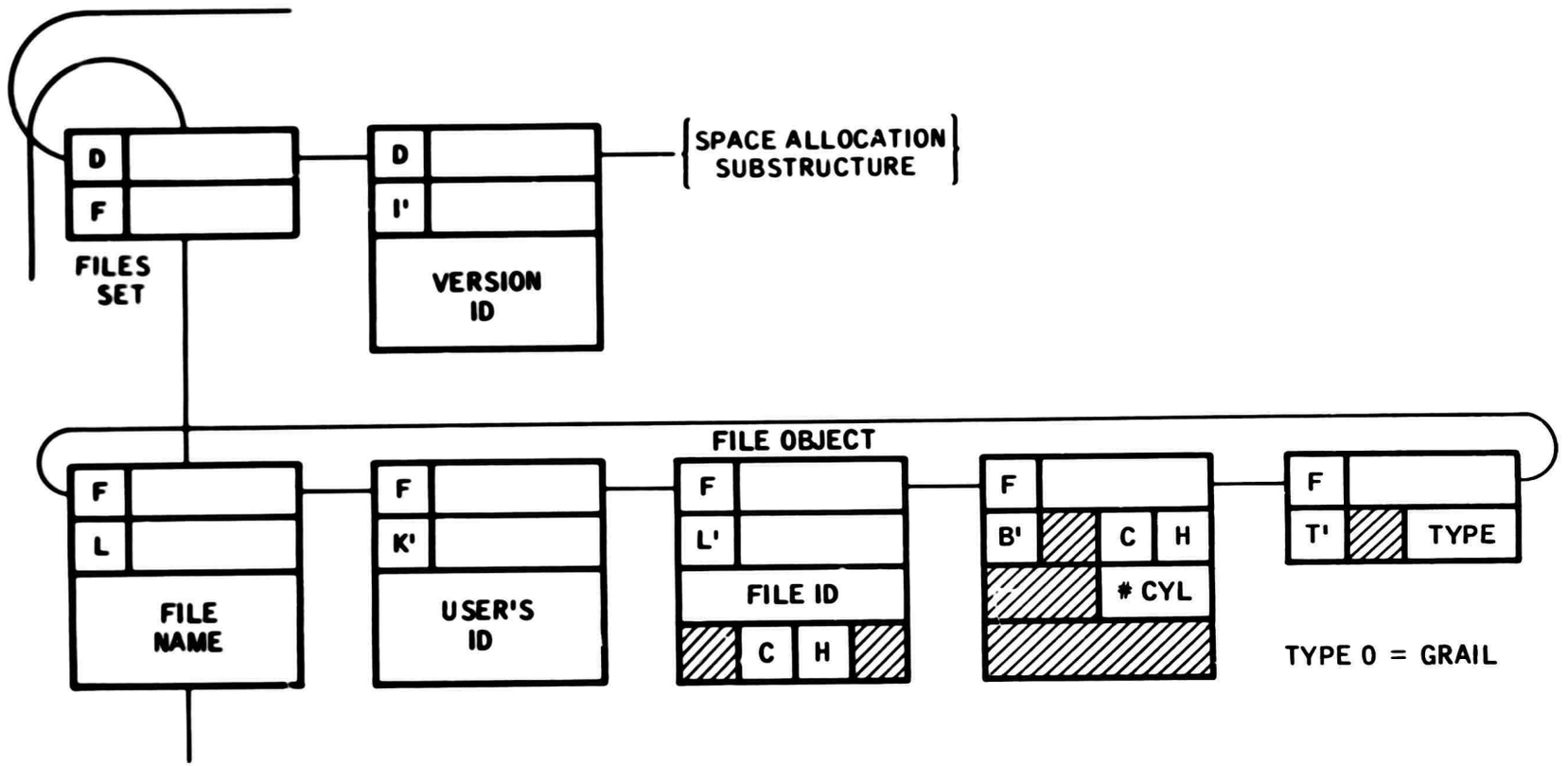


Fig. 8--Files Description Structure

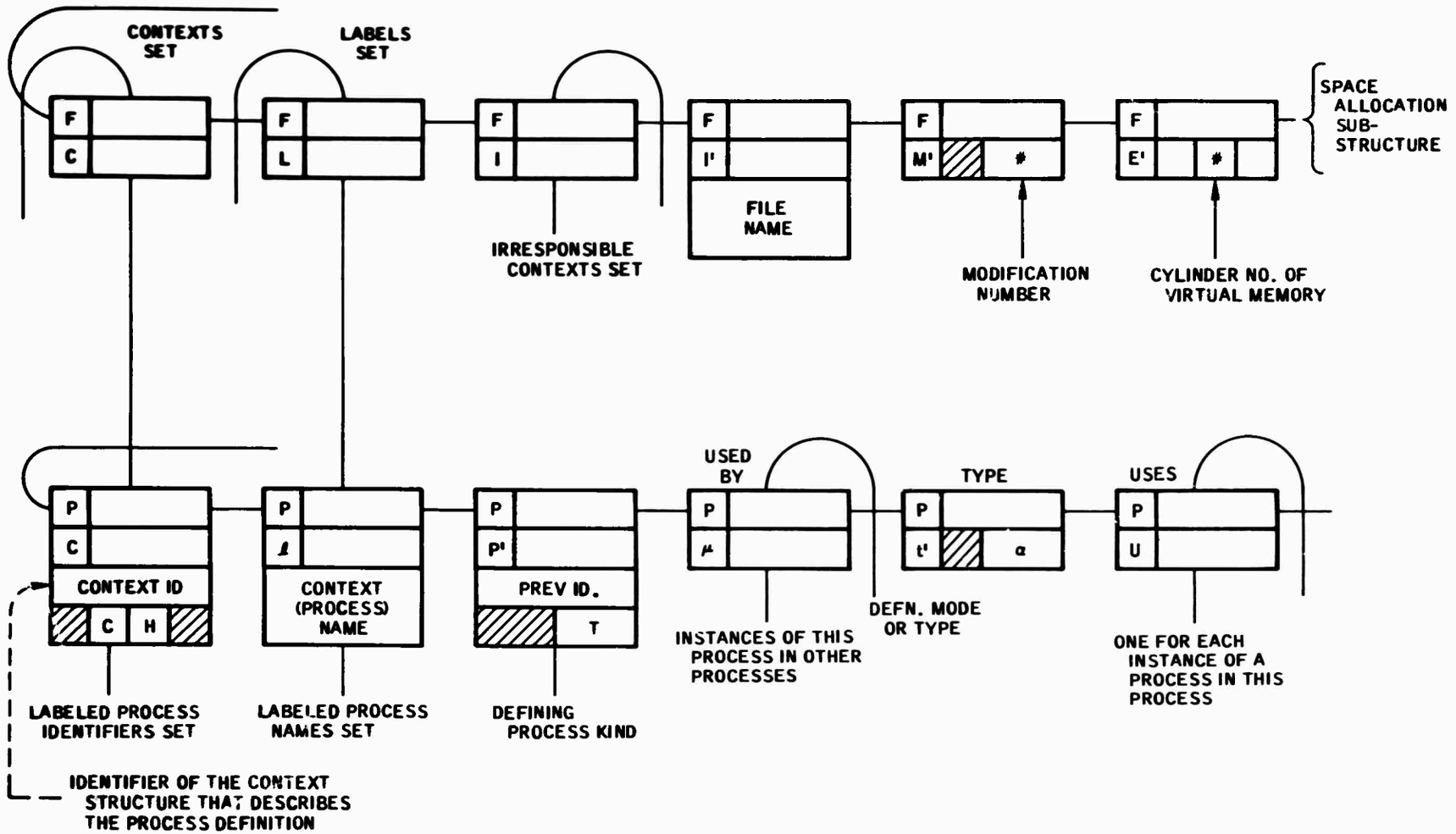


Fig. 9--File Structure

specifies the interrelationship, the name, and the relative location of labeled process definitions within the file.

One cylinder of the file space serves as a virtual memory during interpretive execution of processes. The *occupied* set ring of the space allocation substructure describes only compiled process data sets in the file. The modification number is a counter for the number of times that a file has been rewritten, and thus provides a process with a key as to whether it has the latest information. The file name is user-specified and matches the file name in the FDS. A set of process definition object rings, one object ring for each process in the file, is similar to the file object rings in the FDS. That is, each object ring delineates the attributes of a process definition; in the FDS, each object delineates the attributes of a file.

CONTEXT STRUCTURE

The context structure is part of the internal organization of a labeled process definition. One such structure for each labeled process definition in the user's file resides in the user's file space in secondary storage. A copy of the context structure for the process definition (a frame of which is being displayed) exists in primary storage so that GRAIL can respond to the user's manipulations.

The context structure contains two kinds of information (see Figs. 10 and 11): 1) those labels and their attributes that can be referenced throughout the labeled process definition (others cannot be so referenced because they are restricted to the plane in which they appear), and 2) the hierarchy of open-process (unlabeled) instances or labeled process instances within the labeled process definition.

The label substructure (see Fig. 11) contains labels and their attributes that are addressable through the labeled process definition. This substructure is automatically built from label references whenever the user

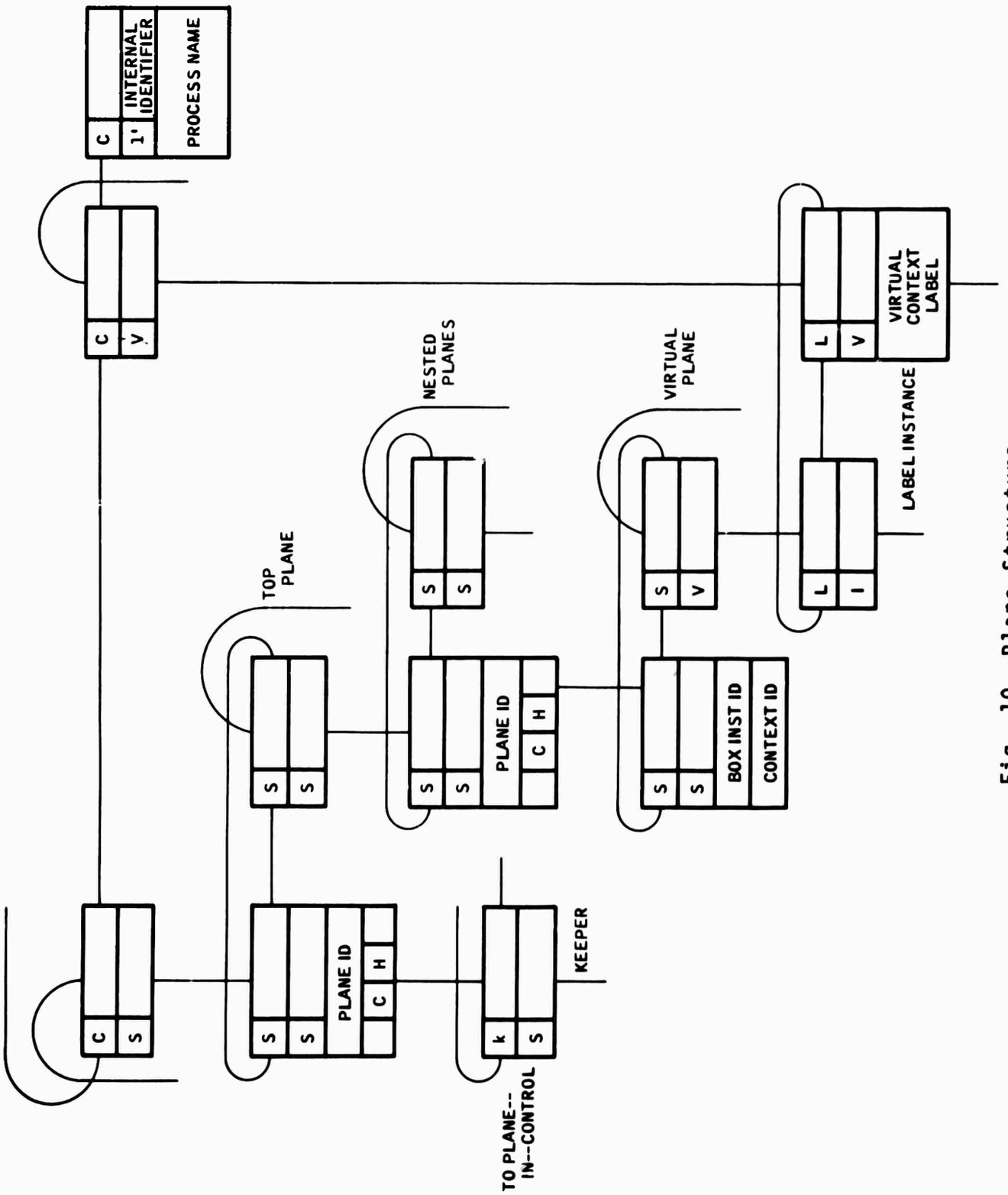


Fig. 10--Plane Structure

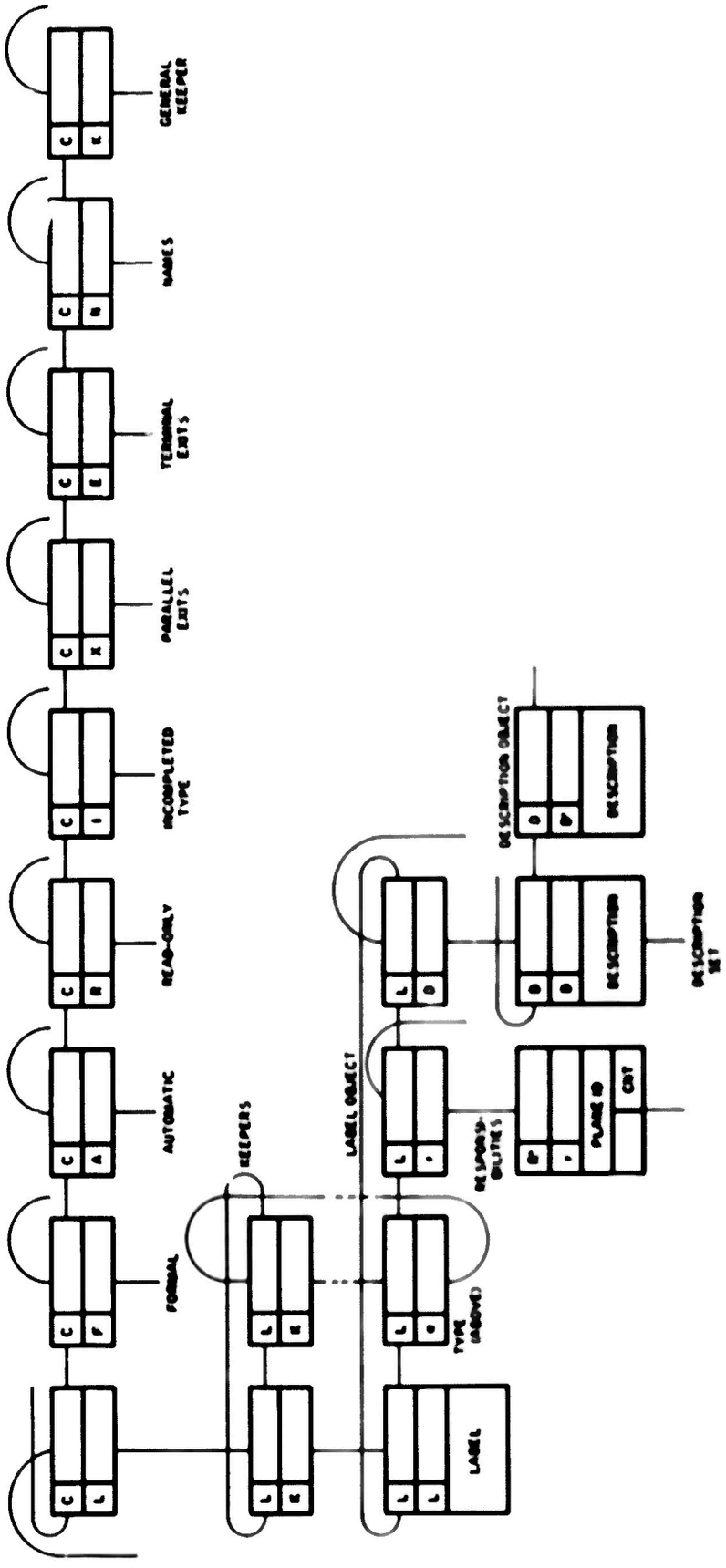


Fig. 11--Context Structure

writes a label in any of the following three places: a flow chart frame, a coding sheet frame, or a data-definition frame. An object ring, one for each label, specifies the label and its attributes. One of its attributes is its type (automatic, formal parameters, etc.). Another is a set of description object rings, one for each line associated with the label on the data-definition frame. For each description line, the user supplies the pseudo-operation code, data declaration, and comments. The number of elements on each description object ring depends on the length of the commentary printed by the user. The three *keepers* (see Fig. 11) serve as place markers when the user edits a statement or graphical symbol referencing a label. A fourth attribute of the label is a responsibility set ring denoting the plane structure data set ID for the plane from which the data were referenced, and a count of its references in that plane.

The process name and its internal identifier also appear in the context structure (see Fig. 10).

The remainder of the structure specifies the hierarchy of open planes in the labeled process definition and the instances of other labeled processes (virtual contexts) in the process definition. In Fig. 10, each S/s set represents the definition of a plane; each S/S represents an instance of an open or virtual process in that plane. If an S/S object-connects to an S/s, it signifies an open-process: if to an S/V, it signifies an instance of a labeled process. The plane structure ID and the secondary storage location (relative to the file base) are given on the S/S elements for open-process definitions (planes) in the labeled process definition.

Instances of other labeled process definitions are virtual within this process. The element (S/S) gives the virtual process context structure ID (the corresponding location [C,H] is found in this file structure), and the

ID of the labeled instance in this plane structure. The *keeper* in the C/S set (see Fig. 10) is object-connected to the plane from which a frame is being displayed. It is used to mark the position of the plane that the user is currently viewing, and changes accordingly as the user views different display frames.

PLANE STRUCTURE

There is at least one plane structure in the secondary storage file space (see Figs. 12-17) for each process definition. A copy of the plane representing the process being operated upon is in primary storage. The plane structure contains information about the flow-line connections between elements in each frame as well as the label translation for each labeled process instance in the frame. The plane structure also contains information that permits an association between the virtual image and its internal logical representation.

Figure 12 shows the basic object ring containing the various attribute sets and the plane-structure data set ID.

Figure 13 shows the frame's structure for frames in the plane. Each frame object specifies:

1) The display frame ID and its location relative to the file base;
2) A source of tag identifiers that become uniquely associated with each graphic element, and that are used with the hardware match circuitry to detect a stylus pointing to the flow lines;
3) A set of CCW objects, one for each figure displayed in the frame.

The Channel Command Word (CCW) object item element connects to a graphic object ring specifying the attributes of the displayed figure. For the frame being displayed, an external element (the PCCW), replaces the CCW displacement element on the CCW object.

Figure 14 shows the translation of formal parameters for a process instance, and also the set of defined decisions.

Figures 15-17 show the graphic object rings and their attributes, e.g., flow connectivity.

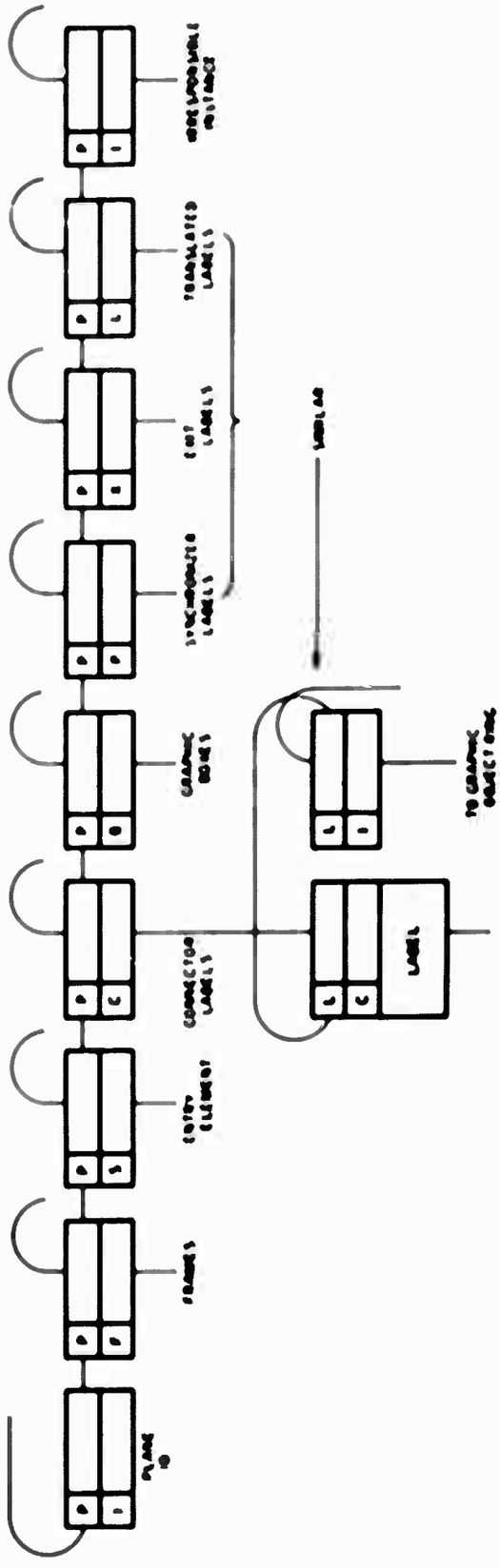


Fig. 12--Plane Structure

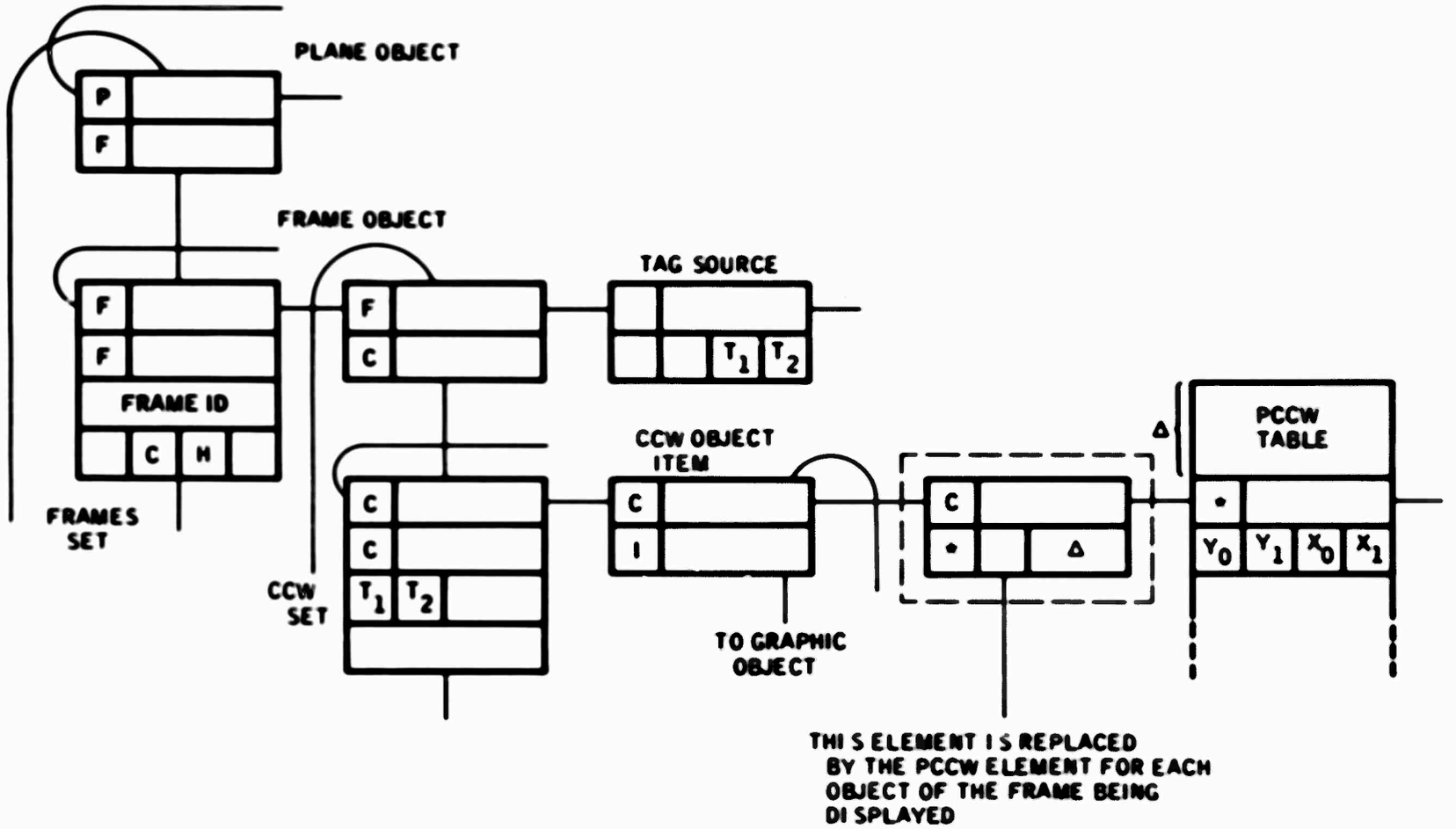


Fig. 13--Plane Structure

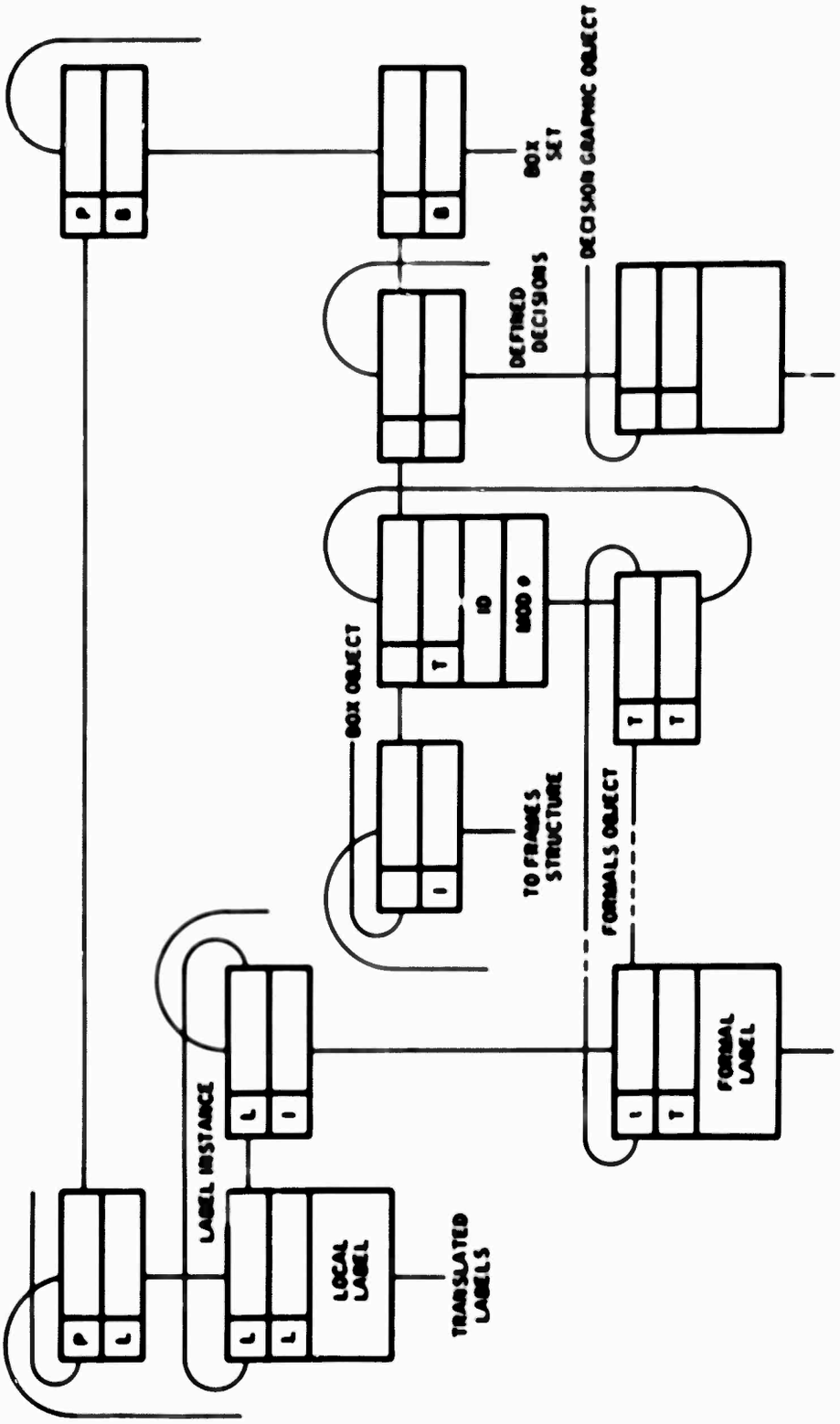
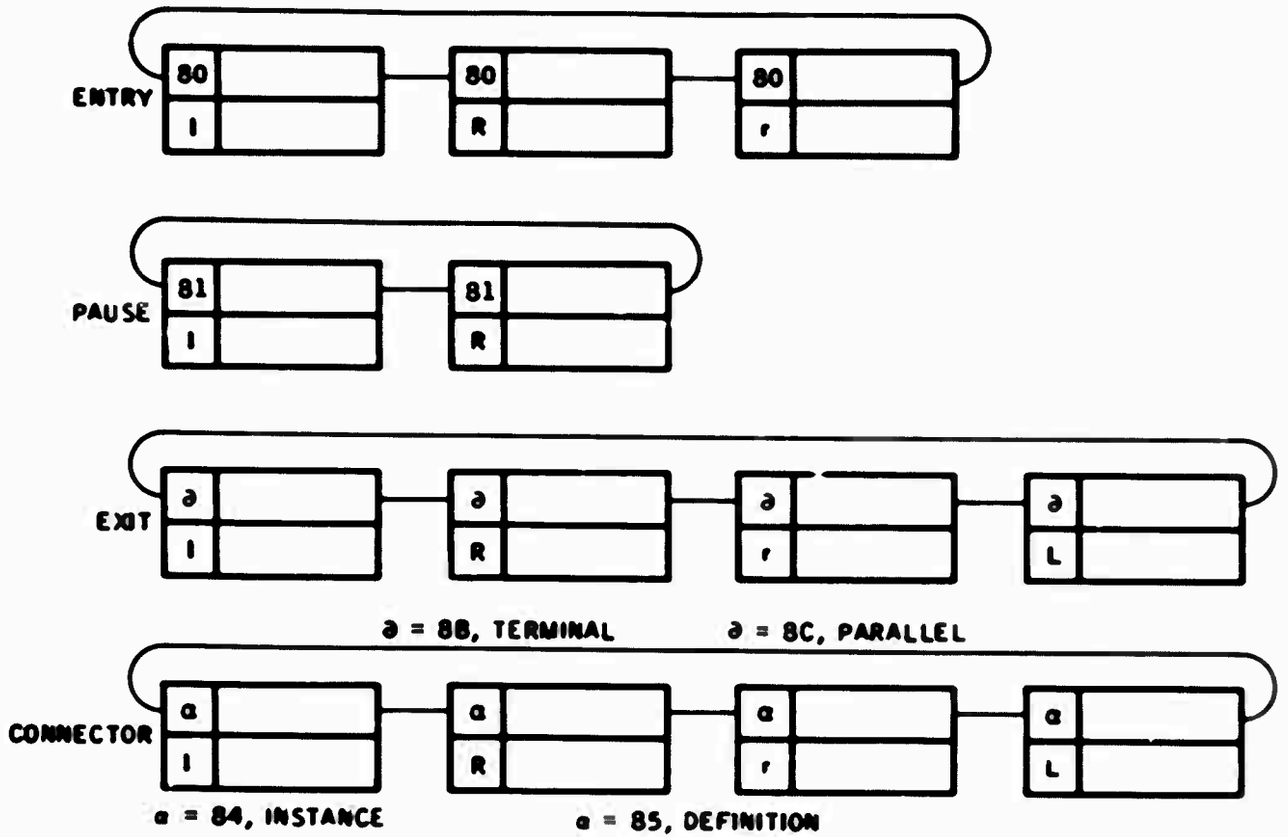


Fig. 14--Plane Structure



LEGEND

- I = ITEM, CONNECTS TO CCW OBJECT
- R = INCOMING FLOW LINE
- r = OUTGOING FLOW LINE
- L = LABEL, CONNECTS TO LABEL INSTANCE SUBSTRUCTURE
- ι = SWITCH INDEX LABEL INSTANCE CONNECTION
- T' = SYNC. TYPE
- I' = BOX ID
- d = DEFINED DECISIONS
- B = BOX SET
- T = FORMAL TRANSLATION

Fig. 15--Plane Structure

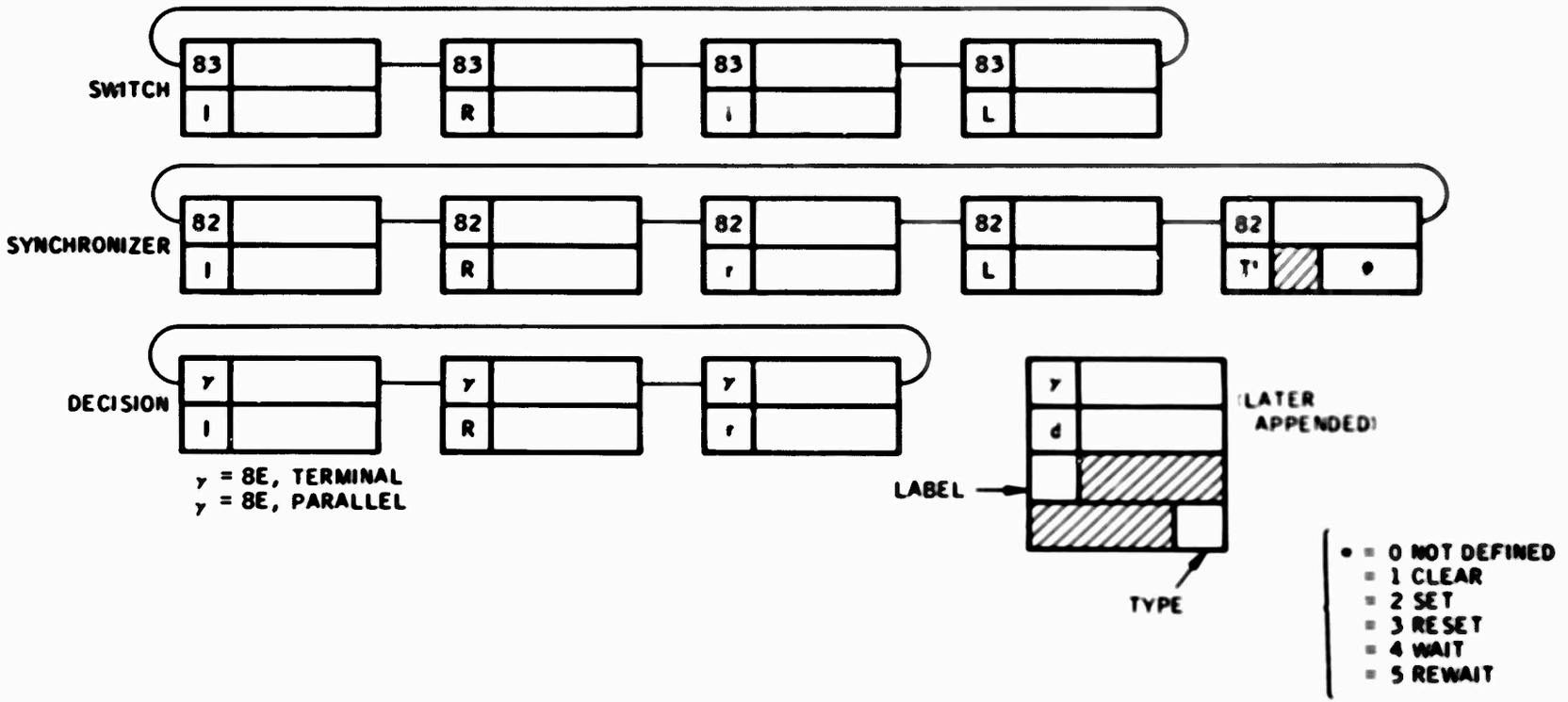


Fig. 16--Plane Structure

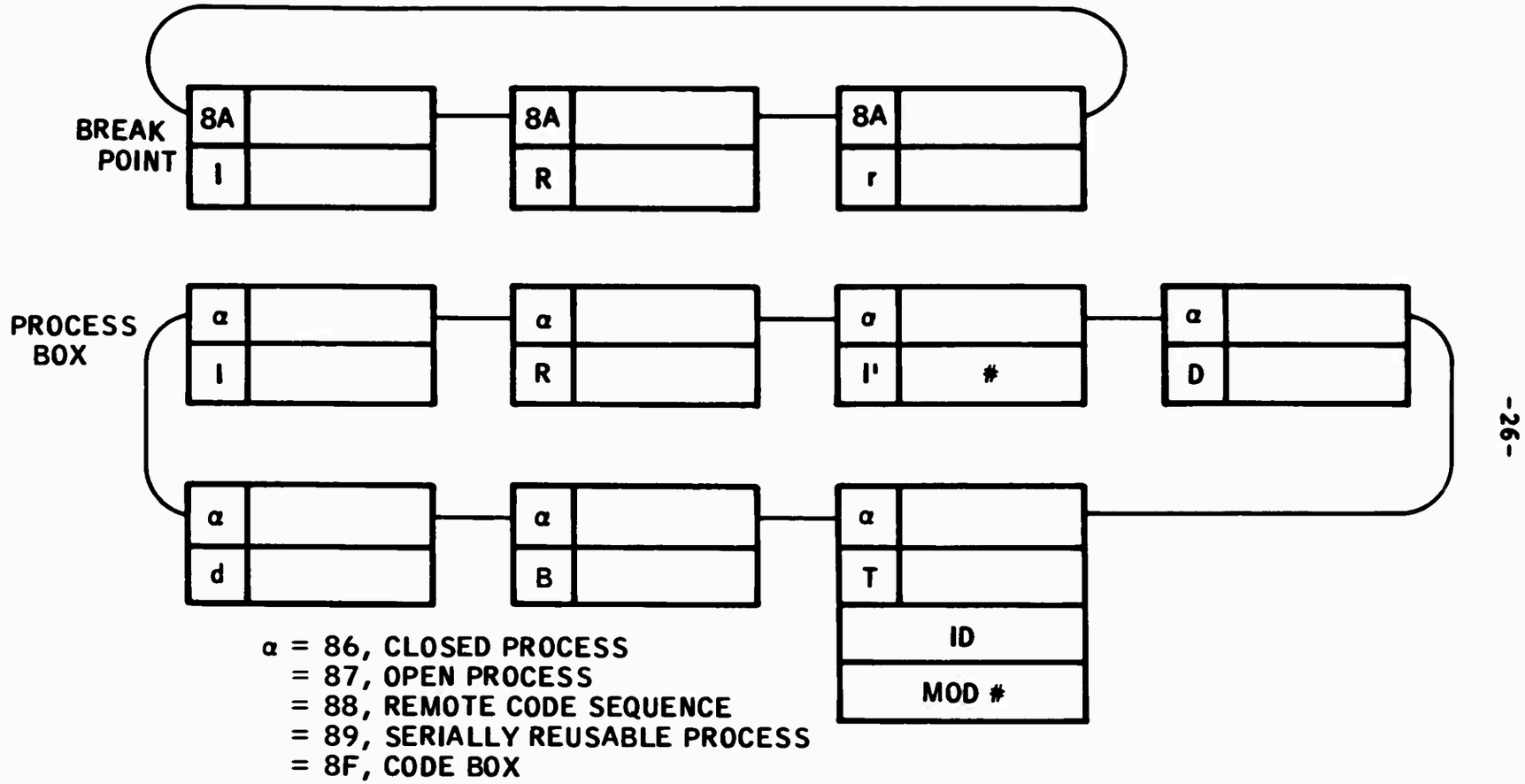


Fig. 17--Plane Structure

III. RING STRUCTURE PRIMITIVES

The primitives are Remote Code Sequences (RCS) that perform the basic operations on all GRAIL ring structures. For example, they find, procure, and release elements. Because the primitives use the same hardware registers for consecutive primitive operations, there is a minimum need for intervening instructions by the invoking processes. Unless the primitive I/O hardware registers contain an output, they remain unchanged over a primitive. The contents of other hardware registers may be destroyed.

FUNCTIONS OF PRIMITIVES

Notes to the following:

1) The address of an element, i.e., A (element), is the address of its first byte, regardless of whether it is object- or set-connected.
2) The address of a ring, i.e., A (ring), is the address of any element belonging to the ring.
3) C (Rα) means content of symbolic register α.

OBTAIN AND RELEASE PRIMITIVES

1) Get an 8-byte element from available space.
   Input C(R6) = A (The address of an array†).
   Output C(R7) = A (The address of an element, if an element is available).
   = 0, if no more element space is available.
2) Get a 16-byte element from available space. The parameters are the same as in the above.

†See Fig. 3 (p. 5).

3) Build an object ring from available space elements.
   Input C(R6) = A (an array).
   C(R7) = A (a LIST).
   LIST contains - α, B1,B2,...BN,00.
   C1,C2,...CK,00, where Bi is the set code byte for the ith 8-byte element and where c_j is the set code byte for the jth 16-byte element, and where the 00 following BN and the 00 following the CK indicate the termination of the 8- and 16-byte elements, respectively. The common object code byte is α.
   Output C(R7) = A (address of a ring).
4) Return element to available space.
   Input C(R6) = A (an array).
   C(R7) = A (element).
   Output - None.
5) Return a ring to available space. This primitive function is a process, not a remote code sequence. It is the single exception.
   Parameters: A (RING), the higher-order bit indicates either object or set ring to be returned.
   A (ARRAY)
   Return.

LOCATE ATTRIBUTE PRIMITIVES

1) Search object ring for the element described by the object and/or set-code bytes.
   Input C(R6) = A (an array).
   C(R7) = A (ring).
   C(R8) = A (object-code byte, set-code byte).
   Output C(R7) = A (element, if found).
   = 0, if no match on codes occurs.

NOTE: If the object- or set-code byte of the input description is zero for this primitive and the following one, the object or set code of the element is accepted as a matched comparison. If both code bytes of the input description are zero, the primitive advances to the next element of the ring and, without actually comparing, assumes a successful comparison on both object and set codes.

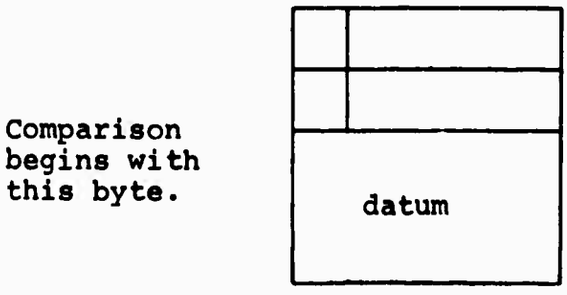
2) Search set ring for the element described by the object- and/or set-code bytes. Parameters are identical to 1) above.

3) Search object ring for the element described by the datum.

Input C(R6) = A (an array).
C(R7) = A (ring).
C(R8) = A (datum to compare).
C(R9) = (# bytes-1) of datum to compare.

Output C(R7) = A (element) if datum is found.
= 0, if no datum is found.

See diagram below:



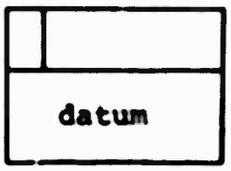
4) Search set ring for the element described by the datum. Parameters are to 3) above.

5) Search object ring for immediate datum.

Input C(R6) = A (an array).
C(R7) = A (ring).
C(R8) = A (4-byte datum to compare).

Output C(R7) = A (element), if datum is found.
= 0, if no datum is found.

The second 4 bytes
are used for comparison
with input datum.

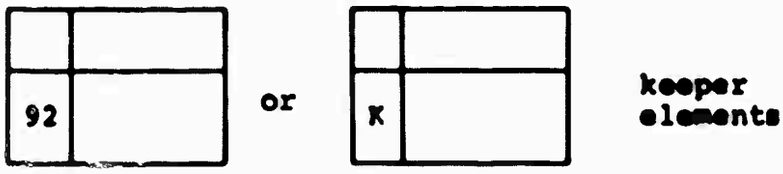


6) Compare link only of set-ring elements.
Input C(R6) = A(an array).
C(R7) = A (ring).
C(R8) = A (4-byte datum, the last 3 bytes
of which will be used for comparison).
See diagram below.
Output C(R7) = A (element), if datum is found.
See diagram below.
0, if no datum is found.

These 3 bytes will be used
for comparison.

These 3 bytes of input data
will be used for comparison.

7) Set ring advance to the nth element. Ignore keeper elements in counting.



Input C(R6) = A (an array).
C(R7) = A (ring).
C(R8) = # elements to advance, zero is illegal.
Output C(R7) = A (nth element from current position).

STRUCTURE MODIFYING PRIMITIVES

1) Insert an element on an object ring.
Input C(R6) = A (an array).
C(R7) = A (element).
C(R8) = A (object ring).
Output - None.
2) Insert an element on a set ring. Parameters are identical to 1) above, except C(R8) = A (set ring).
3) Delete an element from an object ring.
Input C(R6) = A (an array).
C(R7) = A (element).
Output - None.
4) Delete an element from a set ring. Parameters are identical to 3).
5) Merge two object rings.
Input C(R6) = A (an array).
C(R7) = A (first ring).
C(R8) = A (second ring).
Output - None.
6) Merge two set rings. Parameters are identical to 5) above.

REFERENCES

1. Ellis, T. O., J. F. Heafner, and W. L. Sibley, *The GRAIL Project: An Experiment in Man-Machine Communications*, The RAND Corporation, RM-5999-ARPA, September 1969.

2. -----, *The GRAIL Language and Operations*, The RAND Corporation, RM-6001-ARPA, September 1969.

3. -----, *The GRAIL System Implementation*, The RAND Corporation, RM-6002-ARPA, September 1969.

OCUMENT CONTROL DATA

1 ORIGINATING ACTIVITY The Rand Corporation	2a REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b GROUP

3 REPORT TITLE

THE GRAIL RING STRUCTURE AND PRIMITIVES

4 AUTHOR(S) (Last name, first name, initial)

Ellis, T. O., J. F. Heafner and W. L. Sibley

5 REPORT DATE	6a TOTAL NO OF PAGES	6b NO OF REFS
April 1970	42	3

7 CONTRACT OR GRANT NO	8 ORIGINATOR'S REPORT NO
DAC(15-67-C-0141	RM-6241-ARPA

9a AVAILABILITY/LIMITATION NOTICES	9b SPONSORING AGENCY
DDC-1	Advanced Research Projects Agency

10 ABSTRACT

A description of the ring-structure mechanism that controls the disposition of data and storage space in the GRAIL system. The ring structures--continuous chains of elements containing codes and either links or data--represent the logic of both the user's program and of the system itself. The basic types are (1) system structure; (2) files description structure; (3) file structure; (4) context structure; and (5) plane structure. Common to the first three types is a space allocation substructure that describes the location of data sets within a particular area in secondary storage and of space available for use. Basic operations on ring structures--such as finding, pro[illegible]ing, and releasing elements--are performed by ring structure primitives, written as a group of remote code sequences.

11 KEY WORDS

Computer graphics
Computer programming language
GRAIL (Graphical Input Language)